

Chapter 4

Router Configuration

This chapter explains how to use the JUNOScript application programming interface (API) to change router configuration or to request information about the current configuration. The JUNOScript tag elements described here correspond to JUNOS command-line interface (CLI) configuration statements described in the JUNOS Internet software configuration guides.

This chapter discusses the following topics:

Mapping between CLI Configuration Statements and JUNOScript Tag Elements on page 42

Same Tag Elements Used for Requests and Responses on page 48

Overview of Router Configuration Procedures on page 49

Lock the Candidate Configuration on page 50

Request Configuration Information on page 51

Change the Candidate Configuration on page 61

Verify the Syntactic Correctness of the Candidate Configuration on page 72

Commit the Candidate Configuration on page 72

Unlock the Candidate Configuration on page 77

- **Mapping between CLI Configuration Statements and JUNOScript Tag Elements**

The JUNOScript API defines a tag element for every container and leaf statement in the JUNOS configuration hierarchy. At the top levels of the configuration hierarchy, there is almost always a one-to-one mapping between tag elements and statements, and most tag names match the configuration statement name. At deeper levels of the configuration hierarchy, the mapping is sometimes less direct, because some CLI notational conventions do not map directly to Extensible Markup Language (XML)-compliant tagging syntax. The following sections describe the mapping between configuration statements and JUNOScript tag elements:

[Tag Element Mappings for Top-Level \(Container\) Statements on page 42](#)

[Tag Element Mappings for Leaf Statements on page 43](#)

[Tag Element Mappings for Identifiers on page 44](#)

[Tag Element Mappings for Leaf Statements with Multiple Values on page 45](#)

[Tag Element Mappings for Multiple Options on One or More Lines on page 46](#)

[Tag Element Mapping for Comments about Configuration Statements on page 47](#)



Note

For some configuration statements, the notation used when typing the statement at the CLI configuration-mode prompt differs from the notation used in a configuration file. The same JUNOScript tag element maps to both notational styles.

Tag Element Mappings for Top-Level (Container) Statements

The <configuration> tag element is the top-level JUNOScript container tag element for configuration statements, and corresponds to the CLI [edit] level. Most statements at the next few levels of the configuration hierarchy are container statements, and the name of each corresponding JUNOScript container tag element almost always matches the container statement name.



Note

The top-level <configuration-text> tag element also corresponds to the CLI configuration mode's [edit] level. It encloses formatted ASCII configuration statements instead of JUNOScript tag elements, and is not relevant to the following discussion. For more information, see "Specify Formatted ASCII or JUNOScript-Tagged Output" on page 52.

The following depicts the mappings for two sample statements that begin at the top level of the configuration hierarchy. Note how a closing brace in a CLI configuration statement corresponds to a closing JUNOScript tag.

CLI Configuration Statements	JUNOScript Tags
<pre>system { login { ...child statements... } } protocols { ospf { ...child statements... } }</pre>	<pre><configuration> <system> <login> <!- - child statements - -> </login> </system> <protocols> <ospf> <!- - child statements - -> </ospf> </protocols> </configuration></pre>

T1009

Tag Element Mappings for Leaf Statements

A *leaf statement* is a CLI configuration statement that does not contain any other statements. Most leaf statements define a value for one characteristic of a configuration object and have the following form:

keyword *value*;

In general, the name of the JUNOScript tag element corresponding to a leaf statement is the same as the keyword string. The string between the opening and closing JUNOScript tags is the *value*.

The following depicts the mappings for two sample leaf statements that have a keyword and a value: the message statement at the [edit system login] level and the preference statement at the [edit protocols ospf] level:

CLI Configuration Statements	JUNOScript Tags
<pre>system { login { message "Authorized users only"; ...other statements under login ... } } protocols { ospf { preference 15; ...other statements under ospf ... } }</pre>	<pre><configuration> <system> <login> <message>Authorized users only</message> <!- - other children of the <login> tag - -> </login> </system> <protocols> <ospf> <preference>15</preference> <!- - other children of the <ospf> tag - -> </ospf> </protocols> </configuration></pre>

T1010

- Some leaf statements consist of a fixed-form keyword only, without an associated variable-form value. The JUNOScript API represents such statements with an empty tag. The following example shows the mapping for the disable statement at the [edit forwarding-options sampling] hierarchy level:

CLI Configuration Statement	JUNOScript Tags
<pre>forwarding-options { sampling { disable; ...other statements under sampling ... } }</pre>	<pre><configuration> <forwarding-options> <sampling> <disable/> <!-- other children of the <sampling> tag --> </sampling> </forwarding-options> </configuration></pre>

T101

Tag Element Mappings for Identifiers

At some hierarchy levels, the same kind of container object can appear multiple times. Each instance of the object has a unique identifier to distinguish it from the other instances. In the JUNOS CLI notation, the first statement in the set of statements for such an object consists of a keyword and identifier of the following form:

```
keyword identifier {
    ... configuration statements for individual characteristics ...
}
```

The keyword is a fixed string that indicates the type of object being defined, and the *identifier* is the unique name for this instance of the type. In the JUNOScript API, the tag element corresponding to the keyword is a container tag element for child tag elements that represent the object's characteristics. The container tag element's name generally matches the keyword string.

The JUNOScript API differs from the CLI in its treatment of the identifier. Because the JUNOScript API does not allow container tag elements to contain both other tag elements and untagged character data such as an identifier name, the identifier must be enclosed in a tag element of its own. Most frequently, identifier tag elements are called `<name>`. To verify the identifier tag element name for an object, consult the object's entry in the appropriate document type definition (DTD) or in the *JUNOScript API Reference*.

Identifier tag elements also constitute an exception to the general XML convention that tag elements at the same level of hierarchy can appear in any order; the identifier tag element always occurs first within the container tag element.

The [edit protocols bgp group] statement is an example of a configuration statement with an identifier. Each Border Gateway Protocol (BGP) group has an associated name (the identifier) and can have other characteristics such as a type, peer autonomous system (AS) number, and neighbor address.

The following example illustrates the mapping between the CLI statements and JUNOScript tag elements that create two BGP groups called G1 and G2. Notice how the JUNOScript <name> tag element that encloses the identifier for each group (and for the identifier of the neighbor within a group) does not have a counterpart in the CLI statements. For complete information about changing router configuration, see “Change the Candidate Configuration” on page 61.

CLI Configuration Statements	JUNOScript Tags
<pre>protocols { bgp { group G1 { type external; peer-as 56; neighbor 10.0.0.1; } group G2 { type external; peer-as 57; neighbor 10.0.10.1; } } }</pre>	<pre><configuration> <protocols> <bgp> <group> <name>G1</name> <type>external</type> <peer-as>56</peer-as> <neighbor> <name>10.0.0.1</name> </neighbor> </group> <group> <name>G2</name> <type>external</type> <peer-as>57</peer-as> <neighbor> <name>10.0.10.1</name> </neighbor> </group> </bgp> </protocols> </configuration></pre>

T1012

Tag Element Mappings for Leaf Statements with Multiple Values

Some JUNOS CLI leaf statements accept multiple values, which can either be user-defined or drawn from a set of predefined values. CLI notation uses square brackets to enclose all values in a single statement, as in the following:

statement [value1 value2 value3];

The JUNOScript API instead encloses each value in its own tag element. The following example illustrates the mapping between a CLI statement with multiple user-defined values and the corresponding JUNOScript tag elements. The import statement imports two routing policies defined elsewhere in the configuration. For complete information about changing router configuration, see “Change the Candidate Configuration” on page 61.

CLI Configuration Statements	JUNOScript Tags
<pre>protocols { bgp { group 23 { import [policy1 policy2]; } } }</pre>	<pre><configuration> <protocols> <bgp> <group> <name>23</name> <import>policy1</import> <import>policy2</import> </group> </bgp> </protocols> </configuration></pre>

T1013

- The following example illustrates the same mapping for a CLI statement with multiple predefined values. The permissions statement grants three predefined JUNOS permissions to members of the user-accounts login class.

CLI Configuration Statements

```
system {
    login {
        class user-accounts {
            permissions [ configure admin control ];
        }
    }
}
```

JUNOScript Tags

```
<configuration>
    <system>
        <login>
            <class>
                <name>user-accounts</name>
                <permissions>configure</permissions>
                <permissions>admin</permissions>
                <permissions>control</permissions>
            </class>
        </login>
    </system>
</configuration>
```

T1014

Tag Element Mappings for Multiple Options on One or More Lines

For some configuration objects, the configuration file specifies the value for more than one option on a single line, usually for greater legibility and conciseness. In most such cases, the first option identifies the object and does not have a keyword, but later options are paired keywords and values. The JUNOScript API encloses each option in its own tag element. Because the first option has no keyword in the CLI statement, the JUNOScript API assigns a name to its tag element.

The following example illustrates the mapping of a CLI configuration statement that places multiple options on a single line to the corresponding JUNOScript tag elements. Notice that the JUNOScript API defines a tag element for both options and assigns a name to the tag element for the first option (10.0.0.1), which has no CLI keyword.

CLI Configuration Statements

```
system {
    backup-router 10.0.0.1 destination 10.0.0.2;
}
```

JUNOScript Tags

```
<configuration>
    <system>
        <backup-router>
            <address>10.0.0.1</address>
            <destination>10.0.0.2</destination>
        </backup-router>
    </system>
</configuration>
```

T1015

The configuration file entries for some configuration objects have multiple lines with more than one option, and again the JUNOScript API defines a separate tag element for each option. The following example illustrates the mappings for a sample [edit protocols isis traceoptions] statement, which contains three statements, each with multiple options:

CLI Configuration Statements	JUNOScript Tags
<pre> protocols { isis { traceoptions { file trace-file size 3m files 10 world-readable; flag route detail; flag state receive; } } } </pre>	<pre> <configuration> <protocols> <isis> <traceoptions> <file> <filename>trace-file</filename> <size>3m</size> <files>10</files> <world-readable/> </file> <flag> <name>route</name> <detail/> </flag> <flag> <name>state</name> <receive/> </flag> </traceoptions> </isis> </protocols> </configuration> </pre>

T1016

Tag Element Mapping for Comments about Configuration Statements

The JUNOS configuration database can include comments that describe statements in the configuration. In CLI configuration mode, the annotate command specifies the comment to associate with a statement at the current hierarchy level. Comments can also be inserted directly into the ASCII version of the configuration file using a text editor. For more information, see the *JUNOS Internet Software Configuration Guide: Getting Started*.

The JUNOScript API encloses comments about configuration statements in the <junos:comment> tag element. (These comments are different than those described in “XML Comments” on page 12, which are enclosed in the strings <!-- and --> and are automatically discarded by the JUNOScript server.)

Place the <junos:comment> tag element immediately before the tag element that represents the configuration statement to associate with the comment. (If the tag element for the associated statement is omitted, the comment is not recorded in the configuration database.) The comment text string should not include any linebreak characters, but should include one of the two delimiters that indicate a comment in the configuration database: either the # character before the comment or the paired strings /* before the comment and */ after it.

- The following example illustrates how to associate comments with the mappings for two statements in a sample [edit protocols ospf area interface] hierarchy:

CLI Configuration Statements

```
protocols {
    ospf {
        /* New backbone area */

        area 0.0.0.0 {
            # Interface from router sj1 to router sj2

            interface so-0/0/0 {
                hello-interval 5;
            }
        }
    }
}
```

JUNOScript Tags

```
<configuration>
    <protocols>
        <ospf>
            <junos:comment>
                /* New backbone area */
            </junos:comment>
            <area>
                <name>0.0.0.0</name>
                <junos:comment>
                    # Interface from router sj1 to router sj2
                </junos:comment>
                <interface>
                    <name>so-0/0/0</name>
                    <hello-interval>5</hello-interval>
                </interface>
            </area>
        </ospf>
    </protocols>
</configuration>
```

T1048

Same Tag Elements Used for Requests and Responses

The JUNOScript server encloses its response to a configuration request in an `<rpc-reply>` tag element, just as for an operational request. Within the `<rpc-reply>` tag element, it by default returns a JUNOScript-tagged response enclosed in a `<configuration>` tag element. (If the client application requests a formatted ASCII response, the server instead encloses the response in a `<configuration-text>` tag element. For more information, see “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 52.)

Enclosing every JUNOScript-tagged configuration response within a `<configuration>` tag element contrasts with how the server encloses each different kind of operational response in a tag element named for that type of response—for example, the `<chassis-inventory>` tag element for chassis information or the `<interface-information>` tag element for interface information.

The JUNOScript tag elements within the `<configuration>` tag element represent configuration hierarchy levels, configuration objects, and object characteristics, always ordered from higher to deeper levels of the hierarchy. When a client application loads a configuration, it emits the same tag elements in the same order as the JUNOScript server uses when returning configuration information. This consistent representation helps make handling configuration information more straightforward. For instance, the client application can change router configuration by requesting the current configuration, storing the JUNOScript server’s response to a local memory buffer, making changes or applying transformations to the buffered data, and reloading the altered configuration. Because the altered configuration is based on the JUNOScript server’s response, it is certain to be syntactically correct. For more information about changing router configuration, see “Change the Candidate Configuration” on page 61.

Similarly, when a client application requests information about a configuration hierarchy level or object, it uses the same tag elements that the JUNOScript server will return in response. To represent the hierarchy level or object about which it is requesting information, the client application sends a complete stream of tag elements from the top of the configuration hierarchy (represented by the <configuration> tag element) down to the requested level or object. The innermost tag element, which represents the level or object, is either empty or includes the identifier tag element only. The JUNOScript server's response includes the same stream of tag elements, but the tag element for the requested level or object contains all the tag elements that represent the level's child configuration elements or object's characteristics. For more information about requesting configuration information, see “Request Configuration Information” on page 51.

One potential difference between the tag stream in the JUNOScript server's response and one emitted by a client application concerns the use of white space. By XML convention, the JUNOScript server ignores white space in the tag stream it receives. In the stream that it emits, however, the JUNOScript server includes newline characters and extra spaces between tag elements. If a client application writes the response to a file, each tag element appears on its own line, and child tag elements are indented from their parents, both of which enhance readability for users. Client applications can ignore or discard the white space, particularly if they do not write the tag elements to a file for later review. Client applications do not need to include the white space in requests to the JUNOScript server.

Overview of Router Configuration Procedures

To read or change router configuration information, perform the following procedures, which are described in the indicated sections:

1. Establish a connection to the JUNOScript server on the router, as described in “Connect to the JUNOScript Server” on page 18.
2. Open a JUNOScript session, as described in “Start the JUNOScript Session” on page 19.
3. (Optional) Lock the current candidate configuration, as described in “Lock the Candidate Configuration” on page 50. Locking the configuration prevents other users or applications from changing it at the same time.
4. Request or change configuration information, as described in “Request Configuration Information” on page 51 and “Change the Candidate Configuration” on page 61.
5. (Optional) Verify the syntactic correctness of the candidate configuration before attempting to commit it, as described in “Verify the Syntactic Correctness of the Candidate Configuration” on page 72.
6. Commit changes made to the candidate configuration (if appropriate), as described in “Commit the Candidate Configuration” on page 72.
7. Unlock the current configuration if it is locked, as described in “Unlock the Candidate Configuration” on page 77.
8. End the JUNOScript session and close the connection to the router, as described in “End the Session and Close the Connection” on page 31.

- Lock the Candidate Configuration

Before reading or changing router configuration, a client application must establish a connection to the JUNOScript server, as described in “Prerequisites for telnet Connections” on page 17, and open a JUNOScript session, as described in “Start the JUNOScript Session” on page 19.

The application can then emit the `<lock-configuration/>` tag enclosed in an `<rpc>` tag element if it needs to prevent other users or applications from changing the candidate configuration. If it is not important to block changes from other applications, the application can begin requesting or changing configuration information immediately, as described in “Request Configuration Information” on page 51 and “Change the Candidate Configuration” on page 61.

Only one application can hold the lock on the candidate configuration at a time. Other users and applications can still read the configuration while it is locked. The lock persists until either the JUNOScript session ends or the client application emits the `<unlock-configuration/>` tag, as described in “Unlock the Candidate Configuration” on page 77.

If the JUNOScript server successfully locks the configuration, it returns an opening `<rpc-reply>` and closing `</rpc-reply>` tag with nothing between them. If it cannot lock the configuration, it returns an `<xnm:error>` tag element instead. Reasons for the failure include the following:

Another user or application has already locked the candidate configuration. The error message reports the login identity of the user or application.

The candidate configuration includes changes that have not yet been committed. To commit the changes, see “Commit the Candidate Configuration” on page 72. To roll back to a previous version of the configuration (and lose the uncommitted changes), see “Roll Back to a Previous Configuration” on page 71.

In the following example, the client application emits the `<lock-configuration/>` tag after opening the JUNOScript session and exchanging initialization information with the JUNOScript server:

Client Application	JUNOScript Server
---------------------------	--------------------------

```
<rpc>
  <lock-configuration/>
</rpc>
```

```
<rpc-reply xmlns:junos="URL"></rpc-reply>
```

T1003

Automatically Discard Uncommitted Changes

By default, if a client application locks the configuration and does not commit it before the JUNOScript session ends, any uncommitted changes that are made during the session are retained in the candidate configuration. When the candidate configuration is subsequently committed, the leftover changes become part of the committed configuration. This can lead to unexpected results if the user or application performing the subsequent commit is unaware of the leftover changes.

To discard uncommitted changes from the candidate configuration when the JUNOScript session ends before the client application commits the configuration, enclose the `<rollback>` tag element within the `<lock-configuration>` tag element and set the `<rollback>` tag element's value to automatic.

The following example illustrates the sequence of tag elements that causes an automatic rollback:

Client Application	JUNOScript Server
<pre><rpc> <lock-configuration> <rollback>automatic</rollback> </lock-configuration> </rpc></pre>	<pre><rpc-reply xmlns:junos="URL"></rpc-reply></pre>

T1017

Request Configuration Information

To request and parse configuration information, a client application emits the `<get-configuration>` tag element enclosed in an `<rpc>` tag element. By setting optional attributes on the opening `<get-configuration>` tag and enclosing the appropriate child tags within the `<get-configuration>` tag element, the client application can request either the candidate or the committed configuration, specify either JUNOScript-tagged or formatted ASCII output, and request the entire configuration or just a section of it. The following sections describe the procedures for requesting configuration information:

Specify the Committed or Candidate Configuration on page 52

Specify Formatted ASCII or JUNOScript-Tagged Output on page 52

Specify Output Format for Inherited Configuration Groups on page 54

Request the Complete Configuration on page 56

Request One Hierarchy Level on page 57

Request a Single Configuration Object on page 58

If the client application has locked the candidate configuration as described in “Lock the Candidate Configuration” on page 50, it should unlock it after making its read requests. Other users and applications cannot change the configuration while it remains locked. For more information, see “Unlock the Candidate Configuration” on page 77.

- Client applications can also request an XML schema representation of the complete hierarchy of JUNOS configuration statements, as described in the following section:

[Request an XML Schema for the Configuration Hierarchy on page 59](#)

Specify the Committed or Candidate Configuration

To request information from the current committed configuration—the one active on the router—set the database attribute on the opening <get-configuration> tag to the value committed. To request information from the current candidate configuration, either set the database attribute to the value candidate or omit the database attribute completely (the JUNOScript server returns information from the candidate configuration by default). In either case, enclose the <get-configuration> tag element in an <rpc> tag element.

The database attribute can be combined in the opening <get-configuration> tag with the format attribute (described in “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 52), the inherit attribute (described in “Specify Output Format for Inherited Configuration Groups” on page 54), or both. It can be included when requesting either the entire configuration or sections of it (as described in “Request the Complete Configuration” on page 56, “Request One Hierarchy Level” on page 57, and “Request a Single Configuration Object” on page 58).

The following example uses the database attribute to request the entire committed configuration:

Client Application	JUNOScript Server
<pre><rpc> <get-configuration database="committed"/> </rpc></pre>	<pre><rpc-reply xmlns:junos="URL"> <configuration> <version>5.6R1</version> <system> <host-name>big-router</host-name> <! - other children of the <system> tag - > </system> <! - other children of the <configuration> tag - > </configuration> </rpc-reply></pre>

T1050

Specify Formatted ASCII or JUNOScript-Tagged Output

To request that the JUNOScript server return configuration data as formatted ASCII text rather than tagged with JUNOScript tag elements, set the format attribute on the opening <get-configuration> tag to the value text. The server formats its response in the same way as the JUNOS CLI show configuration command displays configuration data—it uses the newline character, tabs, braces, and square brackets to indicate the hierarchical relationships between configuration statements. The server encloses formatted ASCII configuration information in a <configuration-text> tag element.

To request JUNOScript-tagged output, either set the format attribute to the value xml or omit the format attribute completely (the JUNOScript server returns JUNOScript-tagged output by default). The JUNOScript server encloses its output in a <configuration> tag element.

When the JUNOScript server encloses a JUNOScript tag element in the <undocumented> tag element, the corresponding configuration element (hierarchy level or object) is not documented in the JUNOS software configuration guides or officially supported by Juniper Networks. Most often, the undocumented element is used for debugging purposes only by Juniper Networks personnel. In rarer cases, the element is no longer supported or has been moved to another area of the configuration hierarchy, but appears in the current location for backward compatibility.

Regardless of whether they are requesting JUNOScript tag elements or formatted ASCII, client applications must use JUNOScript tag elements to represent the configuration element to display. The format attribute controls the format of only the JUNOScript server's output. Enclose the tag elements that represent the configuration element to display in <get-configuration> and <rpc> tag elements.

The format attribute can be combined in the opening <get-configuration> tag with the database attribute (described in “Specify the Committed or Candidate Configuration” on page 52), the inherit attribute (described in “Specify Output Format for Inherited Configuration Groups” on page 54), or both. It can be included when requesting either the entire configuration or sections of it (as described in “Request the Complete Configuration” on page 56, “Request One Hierarchy Level” on page 57, and “Request a Single Configuration Object” on page 58).

The following example uses the format attribute to request ASCII-formatted output at the [edit policy-options] level of the current candidate configuration:

Client Application	JUNOScript Server
<rpc>	
<get-configuration format="text">	
<configuration>	
<policy-options/>	
</configuration>	
</get-configuration>	
</rpc>	
	<rpc-reply xmlns:junos=" <i>URL</i> ">
	<configuration-text>
	policy-options {
	policy-statement load-balancing-policy {
	from {
	route-filter 192.168.10/24 orlonger;
	route-filter 9.114/16 orlonger;
	}
	then {
	load-balance per-packet;
	}
	}
	</configuration-text>
	</rpc-reply>

T1019

Specify Output Format for Inherited Configuration Groups

The <groups> tag element corresponds to the [edit groups] configuration hierarchy and encloses tag elements representing *configuration groups*, which define sets of configuration statements that can be inserted in multiple locations in the configuration hierarchy by using the apply-groups configuration statement. The section of configuration hierarchy to which a configuration group is applied is said to inherit the group's statements. For more information about configuration groups, see the *JUNOS Internet Software Configuration Guide: Getting Started*.

By default, the JUNOScript server displays the tag element for each configuration group as a child of the <groups> tag element, instead of indicating the inheritance relationships by displaying the tag elements defined in a configuration group as children of the inheriting tag elements. (This parallels the default behavior of the CLI configuration mode show command, which similarly displays the [edit groups] hierarchy as a separate hierarchy in the configuration.)

To request that the JUNOScript server enclose tag elements that are inherited from configuration groups within the tag elements that are inheriting them, and not display the <groups> tag element, set the inherit attribute on the opening <get-configuration> tag to the value inherit. The output is similar to the output from the following CLI configuration mode command:

```
user@host# show | display inheritance | except ##
```

The JUNOScript server encloses its output in <configuration> and <rpc-reply> tag elements.

The inherit attribute can be combined in the opening <get-configuration> tag with the database attribute (described in “Specify the Committed or Candidate Configuration” on page 52), the format attribute (described in “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 52), or both. It can be included when requesting either the entire configuration or sections of it (as described in “Request the Complete Configuration” on page 56, “Request One Hierarchy Level” on page 57, and “Request a Single Configuration Object” on page 58).

For example, consider the following configuration hierarchy, which defines a configuration group called interface-group inherited by the [edit interfaces] hierarchy level:

```
[edit]
groups {
    interface-group {
        interfaces {
            so-1/1/1 {
                encapsulation ppp;
            }
        }
    }
}
apply-groups interface-group;
interfaces {
    fxp0 {
        unit 0 {
            family inet {
                address 192.168.4.207/24;
            }
        }
    }
}
```

When the inherit attribute is not used, the output includes the <groups> tag element enclosing the tag elements defined in the interface-group configuration group. The placement of the <apply-groups> tag element directly above the <interfaces> tag element indicates that the [edit interfaces] hierarchy inherits the statements defined in the interface-group configuration group.

Client Application JUNOScript Server

```
<rpc>
  <get-configuration/>
</rpc>
<rpc-reply xmlns:junos="URL">
  <configuration>
    <groups>
      <name>interface-group</name>
      <interfaces>
        <interface>
          <name>so-1/1/1</name>
          <encapsulation>ppp</encapsulation>
        </interface>
      </interfaces>
    </groups>
    <apply-groups>interface-group</apply-groups>
    <interfaces>
      <interface>
        <name>fxp0</name>
        <unit>
          <name>0</name>
          <family>
            <inet>
              <address>
                <name>192.168.4.207/24</name>
              </address>
            </inet>
          </family>
        </unit>
      </interface>
    </interfaces>
  </configuration>
</rpc-reply>
```

T1052

- When the inherit attribute is used, the output includes the <interfaces> tag element enclosing the tag elements defined by the interface-group configuration group. The <groups> and <apply-groups> tag elements are not displayed.

Client Application	JUNOScript Server
<rpc>	
<get-configuration inherit="inherit"/>	
</rpc>	
	<rpc-reply xmlns:junos=" <i>URL</i> ">
	<configuration>
	<interfaces>
	<interface>
	<name>fxp0</name>
	<unit>
	<name>0</name>
	<family>
	<inet>
	<address>
	<name>192.168.4.207/24</name>
	</address>
	</inet>
	</family>
	</unit>
	</interface>
	<interface>
	<name>so-1/1/1</name>
	<encapsulation>ppp</encapsulation>
	</interface>
	</interfaces>
	</configuration>
	</rpc-reply>

T1053

Request the Complete Configuration

To request the entire current committed or candidate configuration, emit the empty <get-configuration/> tag enclosed in an <rpc> tag element. If desired, include any of the following optional attributes on the <get-configuration/> tag:

- The database attribute, as described in “Specify the Committed or Candidate Configuration” on page 52

- The format attribute, as described in “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 52

- The inherit attribute, as described in “Specify Output Format for Inherited Configuration Groups” on page 54

The following example illustrates the tag sequence for requesting the current candidate configuration tagged with JUNOScript tag elements (the default):

Client Application JUNOScript Server

```
<rpc>
    <get-configuration/>
</rpc>

<rpc-reply xmlns:junos="URL">
    <configuration>
        <version>5.6R1</version>
        <system>
            <host-name>big-router</host-name>
            <!- - other children of the <system> tag - ->
        </system>
        <!- - other children of the <configuration> tag - ->
    </configuration>
</rpc-reply>
```

T1051

Request One Hierarchy Level

To request a single hierarchy level from the current committed or candidate configuration, emit a `<get-configuration>` tag element and enclose the tag elements representing all levels of the configuration hierarchy from the root (represented by the `<configuration>` tag element) down to the level to display. Use an empty tag to represent the requested level. Enclose the entire request in an `<rpc>` tag element.

If desired, include any of the following optional attributes on the opening `<get-configuration>` tag:

The `database` attribute, as described in “Specify the Committed or Candidate Configuration” on page 52

The `format` attribute, as described in “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 52



The `format` attribute controls the format of only the JUNOScript server’s output. Client applications must emit JUNOScript tag elements instead of formatted ASCII to represent which configuration level to display.

The `inherit` attribute, as described in “Specify Output Format for Inherited Configuration Groups” on page 54

- The following example illustrates the tag sequence when a client application requests the contents of the [edit system login] level of the current candidate configuration. The output is tagged with JUNOScript tag elements (the default).

Client Application JUNOScript Server

```
<rpc>
    <get-configuration>
        <configuration>
            <system>
                <login/>
            </system>
        </configuration>
    </get-configuration>
</rpc>
<rpc-reply xmlns:junos="URL">
    <configuration>
        <system>
            <login>
                <user>
                    <name>barbara</name>
                    <full-name>Barbara Anderson</full-name>
                    <!-- other child tags for this user -->
                </user>
                <!-- other children of the <login> tag -->
            </login>
        </system>
    </configuration>
</rpc-reply>
```

T1021

Request a Single Configuration Object

To request information about a single object at a specific level of the configuration hierarchy, emit a `<get-configuration>` tag element enclosing the tag elements representing the entire configuration hierarchy down to the object to display. To represent the requested object, emit its container tag element and identifier tag element only, not any tag elements that represent other characteristics. Enclose the entire request in an `<rpc>` tag element.

If desired, include any of the following optional attributes on the opening `<get-configuration>` tag:

The database attribute, as described in “Specify the Committed or Candidate Configuration” on page 52

The format attribute, as described in “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 52



The format attribute controls the format of only the JUNOScript server’s output. Client applications must emit JUNOScript tag elements instead of formatted ASCII to represent which configuration level to display.

The inherit attribute, as described in “Specify Output Format for Inherited Configuration Groups” on page 54

The following example illustrates the tag sequence when a client application requests the contents of one multicasting scope called local, which is at the [edit routing-options multicast] hierarchy level. To specify the configuration object about which to supply information, the client application emits the <name>local</name> identifier tag element as the innermost tag element. The output is from the current candidate configuration and is tagged with JUNOScript tag elements (the default).

Client Application	JUNOScript Server
<pre><rpc> <get-configuration> <configuration> <routing-options> <multicast> <scope> <name>local</name> </scope> </multicast> </routing-options> </configuration> </get-configuration> </rpc></pre>	<pre><rpc-reply xmlns:junos="URL"> <configuration> <routing-options> <multicast> <scope> <name>local</name> <prefix>239.255.0.0/16</prefix> <interface>ip-f/p/0</interface> </scope> </multicast> </routing-options> </configuration> </rpc-reply></pre>

T1022

Request an XML Schema for the Configuration Hierarchy

To request an XML Schema-language representation of the entire JUNOS configuration hierarchy, emit the <get-xnm-information> tag element enclosing the following two child tag elements with the indicated values:

<type>xml-schema</type>, which specifies that the JUNOScript server return the configuration as an XML schema

<namespace>junos-configuration</namespace>, which specifies that the JUNOScript server return information about the JUNOS configuration

Enclose the entire request in an <rpc> tag element.

The JUNOScript server returns the XML schema enclosed in <rpc-reply> and <xsd:schema> tag elements. The JUNOScript configuration schema represents the entire set of elements that can be configured on a Juniper Networks router that is running the version of the JUNOS software specified by the xmlns:junos attribute in the opening <rpc-reply> tag. Client applications can use the schema to validate the candidate or committed schema on a router, or to discover which configuration statements are available in that version of the JUNOS software.

- The JUNOScript configuration schema does not indicate which elements are actually configured on the router where the JUNOScript server is running, or even that an element can be configured on that type of router (some configuration statements are available only on certain router types). To request the set of currently configured elements and their settings, emit the <get-configuration> tag element instead, as described in other subsections of “Request Configuration Information” on page 51.

Explaining the structure and notational conventions of the XML Schema language is beyond the scope of this document. For a basic introduction to the XML Schema language, see the primer available at <http://www.w3.org/TR/xmlschema-0>. The primer references the more formal specifications for the XML Schema language if you need additional details.

The following examples illustrate the tag sequence with which a client application requests the JUNOScript configuration schema. In the JUNOScript server’s reply, the first two <xsd:element> statements define the schema for the <undocumented> and <comment> JUNOScript tag elements, which can be enclosed in most other container tag elements defined in the schema (container tag elements are defined as <xsd:complexType>).

Client Application	JUNOScript Server
<rpc>	
<get-xnm-information>	
<type>	
xml-schema	
</type>	
<namespace>	
junos-configuration	
</namespace>	
</get-xnm-information>	
</rpc>	
	<rpc-reply xmlns:junos="URL">
	<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" \
	elementFormDefault="qualified">
	<xsd:element name="undocumented">
	<xsd:complexType>
	<xsd:sequence>
	<xsd:any namespace="#any" processContents="skip"/>
	</xsd:sequence>
	</xsd:complexType>
	</xsd:element>
	<xsd:element name="comment">
	<xsd:complexType>
	<xsd:sequence>
	<xsd:any namespace="#any" processContents="skip"/>
	</xsd:sequence>
	</xsd:complexType>
	</xsd:element>
	.
	.

T1023

The third `<xsd:element>` statement in the schema defines the JUNOScript `<configuration>` tag element. The fourth `<xsd:element>` statement begins the definition of the `<system>` tag element, which corresponds to the [edit system] level of the JUNOScript configuration hierarchy. The statements corresponding to other hierarchy levels are omitted for brevity.

Client Application JUNOScript Server

```
</xsd:element>
<xsd:element name="configuration">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="undocumented"/>
        <xsd:element ref="comment"/>
        <xsd:element name="system" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element ref="undocumented"/>
                <xsd:element ref="comment"/>
                <!-- child elements of <system> appear here -->
              </xsd:choice>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <!-- statements for other hierarchy levels appear here -->
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
/rpc-reply>
```

1024

Change the Candidate Configuration

To change the current candidate configuration, emit the `<load-configuration>` tag element enclosed in an `<rpc>` tag element. Specify which configuration element (hierarchy level or configuration object) to change in one of two ways:

By setting the empty `<load-configuration/>` tag's `url` attribute to the pathname of a file that resides on the router and contains the set of configuration statements to load. For example, the following tag identifies the file `/tmp/new.conf` as the file to load:

```
<load-configuration url="/tmp/new.conf"/>
```

By enclosing within the `<load-configuration>` tag element the tag elements that represent the configuration statements to load. Include tag elements for the complete statement path down to the element to change.

The JUNOScript server encloses its response in <rpc-reply> and <load-configuration-results> tag elements. If the load operation succeeds, the <load-configuration-results> tag element encloses the <load-success/> tag. If the load operation fails, the <load-configuration-results> tag element encloses the <load-error-count> tag element, which indicates the number of errors that occurred. In this case, eliminate the errors before committing the candidate configuration.

Client applications can provide the configuration information to load either as formatted ASCII or tagged with JUNOScript tag elements. They can also specify the manner in which the JUNOScript server loads the configuration. For instructions, see the following sections:

- Provide Configuration Data as Formatted ASCII or JUNOScript Tag Elements on page 62
- Merge Statements into the Current Configuration on page 63
- Replace (Override) the Entire Current Configuration on page 64
- Replace a Configuration Element on page 64
- Delete a Configuration Element on page 66
- Change a Configuration Element's Activation State on page 69
- Replace a Configuration Element and Change Its Activation State Simultaneously on page 70
- Roll Back to a Previous Configuration on page 71

Provide Configuration Data as Formatted ASCII or JUNOScript Tag Elements

Client applications can use one of two formats when providing configuration data to be loaded into the candidate configuration:

If providing formatted ASCII (the standard format used by the JUNOS CLI show configuration command to display configuration data), set the format attribute on the opening <load-configuration> tag to the value text. Enclose the configuration data in a <configuration-text> tag element rather than a <configuration> tag element. To indicate the hierarchical relationships between the configuration statements to be loaded, format them using the newline character, tabs and other white space, braces, and square brackets.

If providing JUNOScript-tagged information, either set the format attribute on the opening <load-configuration> tag to the value xml or omit the format attribute completely (the JUNOScript server expects JUNOScript-tagged output by default). Enclose the configuration information in a <configuration> tag element.

Whichever form of configuration information is provided, enclose the <load-configuration> tag element in an <rpc> tag element. The format attribute can be combined with the action attribute, described in “Merge Statements into the Current Configuration” on page 63, “Replace (Override) the Entire Current Configuration” on page 64, and “Replace a Configuration Element” on page 64. For examples of the possible combinations, see those sections.

Merge Statements into the Current Configuration

To combine the statements in the loaded configuration with the current candidate configuration, set the action attribute on the opening <load-configuration> tag to the value merge. This is also the default behavior if there is no action attribute.

```
<load-configuration action="merge">
```

Merging configuration statements is useful when adding a new configuration object or subhierarchy to the configuration. If statements in the loaded configuration conflict with statements in the current candidate configuration, the loaded statements replace the current ones.

As noted in “Provide Configuration Data as Formatted ASCII or JUNOScript Tag Elements” on page 62, client applications can specify the configuration information to load either as formatted ASCII or tagged with JUNOScript tag elements. In the former case, set the format attribute on the opening <load-configuration> tag to text.

The following example merges information for a new interface called so-3/0/0 into the [edit interfaces] level of the current candidate configuration. The information is provided in JUNOScript-tagged format (the default).

Client Application

```
<rpc>
  <load-configuration action="merge">
    <configuration>
      <interfaces>
        <interface>
          <name>so-3/0/0</name>
          <unit>
            <name>0</name>
            <family>
              <inet>
                <address>
                  <name>10.0.0.1/8</name>
                </address>
              </inet>
            </family>
          </unit>
        </interface>
      </interfaces>
    </configuration>
  </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply xmlns:junos="URL">
  <load-configuration-results>
    <load-success/>
  </load-configuration-results>
</rpc-reply>
```

T1054

The following example uses formatted ASCII to define the same new interface:

Client Application	JUNOScript Server
<pre><rpc> <load-configuration action="merge" format="text"> <configuration-text> interfaces { so-3/0/0 { unit 0 { family inet { address 10.0.0.1/8; } } } } </configuration-text> </load-configuration> </rpc></pre>	<pre><rpc-reply xmlns:junos="<i>URL</i>"> <load-configuration-results> <load-success/> </load-configuration-results> </rpc-reply></pre>

T1055

Replace (Override) the Entire Current Configuration

To discard the entire current candidate configuration and replace it with the loaded configuration, set the action attribute on the opening `<load-configuration>` tag to the value `override`:

```
<load-configuration action="override">
```

In the following example, the contents of the file `/tmp/new.conf` (which resides on the router) replaces the entire current candidate configuration. The information in the file is tagged with JUNOScript tag elements (the default), so the `format` attribute is not set.

Client Application	JUNOScript Server
<pre><rpc> <load-configuration action="override" url="/tmp/new.conf"/> </rpc></pre>	<pre><rpc-reply xmlns:junos="<i>URL</i>"> <load-configuration-results> <load-success/> </load-configuration-results> </rpc-reply></pre>

T1056

Replace a Configuration Element

To replace a configuration element (hierarchy level or configuration object) in the current configuration, set the action attribute on the opening `<load-configuration>` tag to the value `replace`:

```
<load-configuration action="replace">
```

If using JUNOScript tag elements to define the loaded configuration, include the tag elements that represent the entire hierarchy down to the configuration element you want to replace. In the opening tag of the container tag element that represents the configuration element, set the replace attribute to the value replace. Within the container tag element, include all its child tag elements.

If using formatted ASCII to define the loaded configuration, include the statements that represent the entire hierarchy down to the element you want to replace. Place the replace: statement above the element's parent statement. Within the container tag elements for the element, include all relevant child statements.

The following example grants new permissions for the object named operator at the [edit system login class] hierarchy level. The information is provided in JUNOScript-tagged format (the default).

Client Application

```
<rpc>
    <load-configuration action="replace">
        <configuration>
            <system>
                <login>
                    <class replace="replace">
                        <name>operator</name>
                        <permissions>configure</permissions>
                        <permissions>admin-control</permissions>
                    </class>
                </login>
            </system>
        </configuration>
    </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply xmlns:junos="URL">
    <load-configuration-results>
        <load-success/>
    </load-configuration-results>
</rpc-reply>
```

T1057

The following example uses formatted ASCII to make the same change:

Client Application

```
<rpc>
    <load-configuration action="replace" format="text">
        <configuration-text>
            system {
                login {
                    replace:
                        class operator {
                            permissions [ configure admin-control ];
                        }
                }
            }
        </configuration-text>
    </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply xmlns:junos="URL">
    <load-configuration-results>
        <load-success/>
    </load-configuration-results>
</rpc-reply>
```

T1058

Delete a Configuration Element

The client application can use the delete attribute to delete several kinds of configuration elements:

- Delete a Hierarchy Level on page 66
- Delete a Configuration Object on page 67
- Delete One or More Values from a Leaf Statement on page 68
- Delete a Fixed-Form Option on page 68



The JUNOS CLI does not provide a delete: statement that marks formatted ASCII configuration data for deletion. Client applications must use JUNOScript tag elements to represent the data to delete.

Delete a Hierarchy Level

To remove a hierarchy level from the configuration hierarchy, set the delete attribute to the value delete on the empty tag that represents the level. Emit a <load-configuration> tag element enclosing the tag elements representing the entire statement path down to the level to remove.

The following example removes the [edit protocols ospf] level of the current candidate configuration:

Client Application	JUNOScript Server
<pre><rpc> <load-configuration> <configuration> <protocols> <ospf delete="delete"/> </protocols> </configuration> </load-configuration> </rpc></pre>	<pre><rpc-reply xmlns:junos="URL"> <load-configuration-results> <load-success/> </load-configuration-results> </rpc-reply></pre>

T1059

Delete a Configuration Object

To delete a single configuration object, set the delete attribute to the value delete on the container tag element for that object. Inside the container tag element, include the identifier tag element only, not any tag elements that represent other characteristics. Emit a <load-configuration> tag enclosing the tag elements representing the entire statement path down to the object to remove.

**Note**

The delete attribute is placed on the opening tag of the containing tag element rather than on the identifier tag element (in the following example, on the <user> tag element rather than the <name> tag element). The presence of the identifier tag element results in the removal of the specified object rather than the removal of the entire hierarchy level represented by the containing tag element (in the example, the user account barbara is removed rather than the entire [edit system login user] level).

The following example removes the account for user object barbara from the [edit system login user] level of the current candidate configuration:

Client Application

```
<rpc>
  <load-configuration>
    <configuration>
      <system>
        <login>
          <user delete="delete">
            <name>barbara</name>
          </user>
        </login>
      </system>
    </configuration>
  </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply xmlns:junos="URL">
  <load-configuration-results>
    <load-success/>
  </load-configuration-results>
</rpc-reply>
```

T1060

- **Delete One or More Values from a Leaf Statement**

To delete one or more values from a leaf statement that accepts multiple values, set the delete attribute to the value delete on the opening tag for each value. Do not include tag elements that represent values to be retained. Emit a <load-configuration> tag element enclosing the tag elements representing the entire statement path down to the leaf statement from which to remove values.

The following example removes two of the permissions granted to the user-accounts login class:

Client Application	JUNOScript Server
<pre><rpc> <load-configuration> <configuration> <system> <login> <class> <name>user-accounts</name> <permissions delete="delete">configure</permissions> <permissions delete="delete">control</permissions> </class> </login> </system> </configuration> </load-configuration> </rpc></pre>	<pre><rpc-reply xmlns:junos="URL"> <load-configuration-results> <load-success/> </load-configuration-results> </rpc-reply></pre>

T1061

- **Delete a Fixed-Form Option**

To delete a fixed-form option, set the delete attribute to the value delete on the tag element for the option. Emit a <load-configuration> tag element enclosing the tag elements that represent the entire statement path down to the option to remove.

The following example removes the fixed-form disable option at the [edit forwarding-options sampling] hierarchy level:

Client Application	JUNOScript Server
<pre><rpc> <load-configuration> <configuration> <forwarding-options> <sampling> <disable delete="delete"/> </sampling> </forwarding-options> </configuration> </load-configuration> </rpc></pre>	<pre><rpc-reply xmlns:junos="URL"> <load-configuration-results> <load-success/> </load-configuration-results> </rpc-reply></pre>

T1062

Change a Configuration Element's Activation State

When a configuration element (hierarchy level or configuration object) is deactivated in the configuration hierarchy, it remains in the candidate configuration but is not activated in the actual configuration when the candidate is later committed.

To use JUNOScript tag elements to define which element to deactivate, either omit the format attribute from the opening `<load-configuration>` tag or set it to the value `xml`. Emit the opening `<configuration>` tag next, and then the tag elements representing the entire statement path down to the element to deactivate. On the tag element that represents the configuration element itself, set the `inactive` attribute to the value `inactive`:

To represent a hierarchy level, emit an empty tag.

To represent an object, emit the object's container tag element. Inside the container tag element enclose only the identifier tag element for the object, not any tag elements that represent other object attributes.

Conclude the tag stream with closing `</configuration>` and `</load-configuration>` tags.

To use formatted ASCII to define the configuration element to deactivate, set the `format` attribute on the opening `<load-configuration>` tag to the value `text`. Emit the opening `<configuration-text>` tag element next, followed by formatted ASCII for all statements in the path down to the element you want to deactivate. Place the `inactive:` statement immediately before the statement for the element. Conclude the tag stream with closing `</configuration-text>` and `</load-configuration>` tags.

To reactivate a configuration element that was previously deactivated, use the preceding instructions and substitute the string `active` for `inactive`. Specifically, when loading JUNOScript tag elements, set the `active` attribute to the value `active` on the opening tag for the element to activate. When loading formatted ASCII statements, place the `active:` statement immediately before the statements to reactivate. With both kinds of notation, the reactivated element is activated in the committed configuration the next time the candidate configuration is committed.

The following example deactivates the [edit protocols isis] level of the current candidate configuration:

Client Application

```
<rpc>
  <load-configuration>
    <configuration>
      <protocols>
        <isis inactive="inactive"/>
      </protocols>
    </configuration>
  </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply xmlns:junos="URL">
  <load-configuration-results>
    <load-success/>
  </load-configuration-results>
</rpc-reply>
```

T1063

Replace a Configuration Element and Change Its Activation State Simultaneously

To replace a configuration element (hierarchy level or configuration object) completely while simultaneously deactivating or reactivating it, set the action attribute on the opening <load-configuration> tag to the value replace. Within the container tag elements for the configuration element you are replacing, include the tag elements or statements that represent all the element's characteristics, not just its identifier tag element or statement. Finally, combine attributes or statements as follows:

If using JUNOScript tag elements to replace and deactivate an element, set two attributes on the opening tag of its container tag element: the replace attribute to the value replace and the inactive attribute to the value inactive. If using formatted ASCII, place the replace: statement above the statements to replace and the inactive: statement directly in front of the first statement.

If using JUNOScript tag elements to replace and reactivate an element, set two attributes on the opening tag of its container tag element: the replace attribute to the value replace and the active attribute to the value active. If using formatted ASCII, place the replace: statement above the statements to replace and the active: statement directly in front of the first statement.

For more information about completely reconfiguring an element, see “Replace a Configuration Element” on page 64.

The following example replaces the information in the [edit protocols bgp] level of the current candidate configuration for the group called G3, but also deactivates the group so that it is not activated in the actual configuration when the candidate is committed:

Client Application

```
<rpc>
  <load-configuration action="replace">
    <configuration>
      <protocols>
        <bgp>
          <group replace="replace" inactive="inactive">
            <name>G3</name>
            <type>external</type>
            <peer-as>58</peer-as>
            <neighbor>
              <name>10.0.20.1</name>
            </neighbor>
          </group>
        </bgp>
      </protocols>
    </configuration>
  </load-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply xmlns:junos="URL">
  <load-configuration-results>
    <load-success/>
  </load-configuration-results>
</rpc-reply>
```

T1064

The following example uses formatted ASCII to make the same change:

Client Application	JUNOScript Server
<pre>rpc> <load-configuration action="replace" format="text"> <configuration-text> protocols { bgp { replace: inactive: group G3 { type external; peer-as 58; neighbor 10.0.20.1; } } } </configuration-text> </load-configuration> /rpc></pre>	<pre><rpc-reply xmlns:junos="URL"> <load-configuration-results> <load-success/> </load-configuration-results> </rpc-reply></pre>
	T1065

Roll Back to a Previous Configuration

The router stores a copy of the most recently committed configuration and up to nine previous configurations. To replace the current candidate configuration with a previous one, set the rollback attribute on the empty <load-configuration/> tag to the numerical index for the appropriate previous configuration. The index for the most recently committed configuration is 0 (zero), and the index for the oldest possible stored configuration is 9.

In the following example, the current candidate configuration is replaced by the most recently committed one:

Client Application	JUNOScript Server
<pre><rpc> <load-configuration rollback="0"/> </rpc></pre>	<pre><rpc-reply xmlns:junos="URL"> <load-configuration-results> <load-success/> </load-configuration-results> </rpc-reply></pre>
	T1066

- Verify the Syntactic Correctness of the Candidate Configuration

Before committing the candidate configuration, you might want to confirm that it is syntactically correct. To verify the candidate configuration without actually committing it, enclose the empty `<check/>` tag in a `<commit-configuration>` tag element.

The JUNOScript server encloses its response in `<rpc-reply>`, `<commit-results>`, and `<routing-engine>` tag elements. If the syntax check succeeds, the `<routing-engine>` tag element encloses the `<commit-check-success/>` tag and the `<name>` tag element, which reports the name of the Routing Engine on which the check succeeded (either re0 or re1). If the syntax check fails, an `<xnm:error>` tag element encloses tag elements that describe the error.

The following example illustrates the tag sequence when the candidate configuration on Routing Engine 0 is syntactically correct:

Client Application	JUNOScript Server
---------------------------	--------------------------

```

<rpc>
    <commit-configuration>
        <check/>
    </commit-configuration>
</rpc>
<rpc-reply xmlns:junos="URL">
    <commit-results>
        <routing-engine>
            <name>re0</name>
            <commit-check-success/>
        </routing-engine>
    </commit-results>
</rpc-reply>

```

T1043

- Commit the Candidate Configuration

To commit the current candidate configuration so that it becomes the active configuration on the router, emit the empty `<commit-configuration/>` tag enclosed in an `<rpc>` tag element. To avoid inadvertently committing changes made by other users or applications, lock the candidate configuration before changing it and emit the `<commit-configuration/>` tag while the configuration is still locked. (For instructions on locking and changing the candidate configuration, see “Lock the Candidate Configuration” on page 50 and “Change the Candidate Configuration” on page 61.) After committing the configuration, unlock it as described in “Unlock the Candidate Configuration” on page 77.

The JUNOScript server encloses its response in `<rpc-reply>`, `<commit-results>`, and `<routing-engine>` tag elements. If the commit operation succeeds, the `<routing-engine>` tag element encloses the `<commit-success/>` tag and the `<name>` tag element, which reports the name of the Routing Engine on which the commit operation succeeded (either re0 or re1). If the commit operation fails, an `<xnm:error>` tag element encloses tag elements that describe the error. The most common causes of failure are semantic or syntactic errors in the candidate configuration.

The following example illustrates the tag sequence when a client application commits the candidate configuration on Routing Engine 0:

Client Application JUNOScript Server

```
<rpc>
    <commit-configuration/>
</rpc>
<rpc-reply xmlns:junos="URL">
    <commit-results>
        <routing-engine>
            <name>re0</name>
            <commit-success/>
        </routing-engine>
    </commit-results>
</rpc-reply>
```

T1044

For information about using JUNOScript for other commit operations, see the following sections:

[Commit and Synchronize the Configuration on Both Routing Engines on page 73](#)

[Commit the Configuration at a Specified Time on page 74](#)

[Commit a Configuration but Require Confirmation on page 76](#)

For more detailed information about commit operations, including a discussion of the interaction among different commit operations, see the *JUNOS Internet Software Configuration Guide: Getting Started*.

Commit and Synchronize the Configuration on Both Routing Engines

To copy the candidate configuration stored on one of a router's Routing Engines to the other Routing Engine, verify the candidate's syntactic correctness, and commit it on both Routing Engines, emit the empty `<synchronize/>` tag enclosed in `<commit-configuration>` and `<rpc>` tag elements. This operation is valid only on a router with more than one Routing Engine installed (the JUNOScript server returns an error if there is only one Routing Engine). Also, the apply groups `re0` and `re1` must already be defined on the router. For more information, see the *JUNOS Internet Software Configuration Guide: Getting Started*.

The JUNOScript server encloses its response in `<rpc-reply>` and `<commit-results>` tag elements. It emits a separate `<routing-engine>` tag element for each operation on each Routing Engine:

If the syntax check succeeds on a Routing Engine, the `<routing-engine>` tag element encloses the `<commit-check-success/>` tag and the `<name>` tag element, which reports the name of the Routing Engine on which the check succeeded (either `re0` or `re1`). If the configuration is incorrect, an `<xnm:error>` tag element encloses tag elements that describe the error.

If the commit operation succeeds on a Routing Engine, the `<routing-engine>` tag element encloses the `<commit-success/>` tag and the `<name>` tag element, which reports the name of the Routing Engine on which the commit operation succeeded. If the commit operation fails, an `<xnm:error>` tag element encloses tag elements that describe the error. The most common causes of failure are semantic or syntactic errors in the candidate configuration.

- The following example illustrates the tag sequence when a client application commits and synchronizes the candidate configuration:

Client Application	JUNOScript Server
<rpc>	
<commit-configuration>	
<synchronize/>	
</commit-configuration>	
</rpc>	
	<rpc-reply xmlns:junos=" <i>URL</i> ">
	<commit-results>
	<routing-engine>
	<name>re0</name>
	<commit-check-success/>
	</routing-engine>
	<routing-engine>
	<name>re1</name>
	<commit-check-success/>
	</routing-engine>
	<routing-engine>
	<name>re0</name>
	<commit-success/>
	</routing-engine>
	<routing-engine>
	<name>re1</name>
	<commit-success/>
	</routing-engine>
	</commit-results>
	</rpc-reply>

T1046

Commit the Configuration at a Specified Time

To commit the candidate configuration at a specified time in the future, emit the <at-time> tag element enclosed in <commit-configuration> and <rpc> tag elements. The configuration is checked immediately for syntactic correctness. If the syntax check succeeds, the JUNOScript server returns <rpc-reply>, <commit-results>, and <routing-engine> tag elements enclosing the <commit-check-success/> tag and the <name> tag element, which reports the name of the Routing Engine on which the check succeeded (either re0 or re1).

The configuration is scheduled for commit at the specified time. The JUNOScript server does not emit additional tag elements when the commit operation is actually performed.

If the configuration is not syntactically correct, an <xnm:error> tag element encloses tag elements that describe the error. The commit operation is not scheduled.

To indicate when to perform the commit operation, include one of three types of values in the <at-time> tag element:

- The string reboot, to commit the configuration the next time the router reboots.

- A time value of the form *hh:mm[:ss]* (hours, minutes, and optionally seconds), to commit the configuration at the specified time, which must be in the future at the time the client application emits the <commit-configuration> tag element but before 11:59:59 PM on the day it emits the tag element. As an example, if the <at-time> tag element specifies 02:00 AM and the application emits the <commit-configuration> tag element at 2:10 AM, the commit will not take place at all because the scheduled time has already passed for that day. The commit is not scheduled for 2:00 AM the next day.

- Use 24-hour time; for example, 04:30:00 means 4:30:00 AM and 20:00 means 8:00 PM. The time is interpreted with respect to the clock and time zone settings on the router.

- A date and time value of the form *yyyy-mm-dd hh:mm[:ss]* (year, month, date, hours, minutes, and optionally seconds), to commit the configuration at the specified day and time, which must be after the <commit-configuration> tag element is emitted. Use 24-hour time. For example, 2003-08-21 12:30:00 means 12:30 PM on August 21, 2003. The time is interpreted with respect to the clock and time zone settings on the router.

The following example illustrates the tag sequence when a client application schedules a commit operation for 10:00 PM on the current day:

Client Application

```
<rpc>
  <commit-configuration>
    <at-time>22:00</at-time>
  </commit-configuration>
</rpc>
```

JUNOScript Server

```
<rpc-reply xmlns:junos="URL">
  <commit-results>
    <routing-engine>
      <name>re1</name>
      <commit-check-success/>
    </routing-engine>
  </commit-results>
</rpc-reply>
```

T1047

Commit a Configuration but Require Confirmation

To commit the current candidate configuration but require an explicit confirmation for the commit to become permanent, emit the empty <confirmed/> tag enclosed in <commit-configuration> and <rpc> tag elements. If the commit is not confirmed within a certain amount of time (10 minutes by default), the JUNOScript server automatically rolls back to the previously committed configuration. To specify a different number of minutes for the rollback deadline, also emit the <confirm-timeout> tag element enclosing a positive integer value.

The JUNOScript server encloses its response in <rpc-reply>, <commit-results>, and <routing-engine> tag elements. If the commit operation succeeds, the <routing-engine> tag element encloses the <commit-success/> tag and the <name> tag element, which reports the name of the Routing Engine on which the commit operation succeeded (either re0 or re1). If the commit operation fails, an <xnm:error> tag element encloses tag elements that describe the error. The most common causes of failure are semantic or syntactic errors in the candidate configuration.

The <confirmed/> tag is useful for verifying that a configuration change works correctly and does not prevent management access to the router. If the change prevents access or causes other errors, the automatic rollback to the previous configuration restores access after the rollback deadline passes.

To delay the rollback to a time later than the current rollback deadline, emit the <confirmed/> tag again (enclosed in a <commit-configuration> tag element) before the deadline passes. Optionally, include the <confirm-timeout> tag element to specify how long to delay the next rollback; omit that tag element to delay the rollback by the default 10 minutes. The client application can delay the rollback indefinitely by emitting the <confirmed/> tag repeatedly in this way.

To cancel the rollback completely (and commit a configuration permanently), emit one of the following tag sequences before the rollback deadline passes:

The empty <commit-configuration/> tag enclosed in an <rpc> tag element. The rollback is cancelled and the current candidate configuration is committed immediately, as described in “Commit the Candidate Configuration” on page 72. If the candidate configuration is still the same as the temporarily committed configuration, this effectively recommits the temporarily committed configuration.

The <commit-synchronize/> tag enclosed in <commit-configuration> and <rpc> tag elements. The rollback is cancelled and the current candidate configuration is checked and committed immediately on both Routing Engines, as described in “Commit and Synchronize the Configuration on Both Routing Engines” on page 73. If a confirmed commit operation has been performed on both Routing Engines, then emitting the <commit-synchronize/> tag cancels the rollback on both.

The <commit-at> tag element enclosed in <commit-configuration> and <rpc> tag elements. The rollback is cancelled and the configuration is checked immediately for syntactic correctness, then committed at the scheduled time, as described in “Commit the Configuration at a Specified Time” on page 74.

The following example illustrates the tag sequence when a client application commits the candidate configuration on Routing Engine 1 with a rollback deadline of 20 minutes:

Client Application	JUNOScript Server
<pre><rpc> <commit-configuration> <confirmed/> <confirm-timeout>20</confirm-timeout> </commit-configuration> </rpc></pre>	<pre><rpc-reply xmlns:junos="URL"> <commit-results> <routing-engine> <name>re1</name> <commit-success/> </routing-engine> </commit-results> </rpc-reply></pre>

T1045

Unlock the Candidate Configuration

To unlock the candidate configuration after changing it, committing it, or both, emit the empty `<unlock-configuration/>` tag enclosed in an `<rpc>` tag element. Other applications and users cannot change the candidate configuration until the client application releases the lock. To confirm that it has successfully unlocked the configuration, the JUNOScript server returns an opening `<rpc-reply>` and closing `</rpc-reply>` tag with nothing between them. If it cannot unlock the configuration, it returns an `<xnm:error>` tag element instead.

The following example illustrates the tag sequence with which the client application unlocks the configuration:

Client Application	JUNOScript Server
<pre><rpc> <unlock-configuration/> </rpc></pre>	<pre><rpc-reply xmlns:junos="URL"></rpc-reply></pre>

T1041

